Reliability and Security of AI Hardware

Dennis Gnad, Jonas Krautter, Angeliki Kritikakou, Vincent Meyers, Paolo Rech, Josie Esteban Rodriguez Condia, Annachiara Ruospo, Ernesto Sanchez, Fernando Fernandes dos Santos, Olivier Sentieys, et al.

▶ **To cite this version:**

Dennis Gnad, Jonas Krautter, Angeliki Kritikakou, Vincent Meyers, Paolo Rech, et al.. Reliability and Security of AI Hardware. ETS 2024 - 29th IEEE European Test Symposium, May 2024, The Hague, Netherlands. pp.1-10. hal-04577494

HAL Id: hal-04577494
https://hal.science/hal-04577494v1

Submitted on 16 May 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Reliability and Security of AI Hardware

Dennis Gnad[1], Martin Gotthard[1], Jonas Krautter[1], Angeliki Kritikakou[2], Vincent Meyers[1], Paolo Rech[3], Josie E. Rodriguez Condia[4], Annachiara Ruospo[4], Ernesto Sanchez[4], Fernando Fernandes dos Santos[2], Olivier Sentieys[2], Mehdi Tahoori[1], Russell Tessier[5], Marcello Traiola[2]
[1]Karlsruhe Institute of Technology, Germany – [2]Inria, University of Rennes, France – [3]University of Trento, Italy
[4]Politecnico di Torino, Italy – [5]University of Massachusetts Amherst, United States
[1]{dennis.gnad, jonas.krautter, vincent.meyers, mehdi.tahoori}@kit.edu, martin.gotthard@maclinwi.de
[2]{fernando.fernandes-dos-santos, angeliki.kritikakou, olivier.sentieys, marcello.traiola}@inria.fr
[3]paolo.rech@unitn.it – [4]{josie.rodriguez, annachiara.ruospo, ernesto.sanchez}@polito.it – [5]tessier@umass.edu

***Abstract*—In recent years, Artificial Intelligence (AI) systems have achieved revolutionary capabilities, providing intelligent solutions that surpass human skills in many cases. However, such capabilities come with power-hungry computation workloads. Therefore, the implementation of hardware acceleration becomes as fundamental as the software design to improve energy efficiency, silicon area, and latency of AI systems. Thus, innovative hardware platforms, architectures, and compiler-level approaches have been used to accelerate AI workloads. Crucially, innovative AI acceleration platforms are being adopted in application domains for which dependability must be paramount, such as autonomous driving, healthcare, banking, space exploration, and industry 4.0. Unfortunately, the complexity of both AI software and hardware makes the dependability evaluation and improvement extremely challenging. Studies have been conducted on both the security and reliability of AI systems, such as vulnerability assessments and countermeasures to random faults and analysis for side-channel attacks. This paper describes and discusses various reliability and security threats in AI systems, and presents representative case studies along with corresponding efficient countermeasures.**

***Index Terms*—AI accelerators, machine learning, neural networks, hardware reliability, fault tolerance, hardware security, side-channel attacks**

I. Introduction

The high number of operations and parameters in modern Deep Neural Networks (DNNs), i.e., hundreds of matrix multiplications per layer and up to trillions of parameters, requires complex and high-performance parallel AI accelerators. Graphics Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs), or specific hardware, such as low-power EdgeAI devices and Tensor Processing Units (TPUs), are examples of devices used to accelerate the training and inference of neural networks. These platforms generally support the efficient implementation of algorithms to deploy Machine Learning (ML) workloads, such as General Matrix Multiplications (GEMM) [1]. In particular, modern devices resort to specialized architectures to speed up the execution of ML workloads. GPUs resort to specialized in-chip ML accelerators (i.e., Tensor Core Units – TCUs) to improve performance and efficiency in the execution of convolution, as GEMM operations in ML workloads. These TCU units are included as part of the GPU's hierarchy, inside the parallel symmetric streaming multiprocessors cores, and resort to specialized scheduling strategies (combining software and hardware) to speed up the execution of small fragments (tiles) from large GEMM operations [2]. FPGAs can provide excellent performance per watt for artificial neural networks [3]. Since they are rather expensive, large companies like Google, Amazon Web Services and Microsoft, provide access to FPGA hardware through the cloud, similar to a GPU [4], or provide ML as a Service (MLaaS) [5]–[7]. In addition to that, multi-tenancy on FPGAs is of great interest for maximizing resource utilization.

Despite having a very different architecture and programming framework, all AI hardware platforms have in common the capability of executing several operations or tasks in parallel. Parallelism has unquestionable benefits in terms of efficiency. However, it introduces three significant challenges when reliability is concerned: (1) Due to the large area and the high amount of resources available, the radiation-induced error rate of the available commercial off-the-shelf (COTS) products for AI is very high [8], [9]. (2) The corruption of critical or shared resources impacts several parallel processes, increasing the chance of radiation-induced misdetections [10]–[12]. (3) The complexity of the architecture makes the design and/or operation overhead necessary to create an intrinsically radiation-tolerant chip too costly for parallel accelerators. In this scenario, it is fundamental to perform experiments to evaluate if the available COTS devices are sufficiently reliable for being adopted as part of safety-critical applications. If the error rate is found to be too high, the design of effective and efficient hardening solutions for DNNs is necessary.

While reliability is related to the protection of the system from unintended faults, security focuses on protecting systems from intentional threats, such as malicious attacks, that may recover secret assets and jeopardize privacy (data theft). This is achieved through *side-channel analysis attacks*, where valuable information can leak and be retrieved by an attacker spying on the system. Recent studies have shown that this kind of attack vector is indeed exploitable to obtain information about the ML algorithm, architecture, and data deployed in a device [13]–[16]. These attacks are performed

remotely without physical access to the device, which makes them even more dangerous.

While AI accelerator platforms offer high performance and efficiency, they are still not well known and explored from a dependability point of view. Therefore, examining early their reliability and security aspects is crucial in identifying potential issues, that require to be solved before being deployed in mission-critical systems. In this paper, we describe and discuss such main challenges, along with possible solutions.

The paper is organized as follows. Section II first discusses existing reliability assessment approaches for AI accelerators, then focuses more in detail on reliability threats and protections for GPUs, and finally the evaluation and mitigation of radiation-induced transient faults are showcased. Section III discusses remote side-channel attacks on FPGA-based neural networks accelerators and possible countermeasures; an image extraction attack and countermeasure are presented. Finally, Section IV concludes the paper.

II. Current Challenges and Solutions for Assessing the Reliability of Hardware Platforms for Machine Learning

Several studies have demonstrated that the latest transistor technologies in modern devices are prone to increase faults during operative stages (i.e., silent data errors [17]) due to internal defects, such as electromigration, premature aging, and wear-out, as well as external impacts from harsh environmental conditions, such as the exposure to high temperature or radiation [18]. Faults may corrupt the ML model prediction drastically [8], [19]. Unfortunately, the fault-induced misprediction probability can be so high as to impede a safe deployment of DNN models at scale, creating a need for efficient and effective hardening solutions. Several beam experiments have been conducted to characterize the radiation response to various natural particles of AI accelerators [10], [20]–[22]. These technology vulnerabilities in combination with the structural parallelism and programming complexity of modern GPUs and their applications (i.e., CNNs) impede the efficient use of traditional methods of reliability evaluation and demand the development or adoption of more effective reliability assessment methods.

A. Methods for reliability assessment of AI accelerators

The reliability of ML hardware is a critical concern for several domains. Thus, efficient and adequate methods for resilience characterization are crucial for the effective identification of vulnerable structures as well as the proposal of countermeasures [23]. These methods can be organized in six main strategies:

1) ***formal evaluations****:* that consist of analytical and theoretical evaluations of the algorithms implementing an ML application [24]. These analyses might include the general features of the structures in a system. However, these hardware-agnostic analyses are far from the reliability assessment of specific components, such as the underlying hardware accelerator used to deploy the system.

2) ***application-based evaluations****:* that focuses on the model's architecture of an ML application, such as corruption on neurons and functions. These evaluations resort to software corruptions (i.e, bit-flip) on data path elements, including inputs, weights, and feature maps among the channels of an application [25]. Unfortunately, these evaluations neglect the workload's distribution on a device and the effects of errors produced by hardware faults on its underlying hardware. Moreover, these analyses can represent, at most, corruption effects from memories in hardware.

3) ***software-based assessment****:* that consist of the instrumentation of the application's code with functions to represent effects on the running instructions from corruption due to faults in the structures of a device [26]. These evaluations use real devices and might accurately describe corruption effects on an application from hardware faults. However, the code instrumentation requires special frameworks and demands considerable engineering effort to ensure the intended functionality. Similarly, the evaluation accuracy directly depends on the definition of the error functions from hardware faults. Moreover, the analyses might be limited to data-path structures in a GPU (e.g., register files, memories, and functional units), and more recently some controllers [27].

4) ***structural and architectural evaluation****:* that consist of functional or fine-grain evaluations of the ML hardware by resorting to structural models or low-level micro-architecture models of a device (at RT- or gate-level). The low-level micro-architecture evaluations resort to simulation (e.g., using logic simulators and frameworks) and emulation-based (e.g., using FPGA platforms) schemes and provide high accuracy in the resilience evaluation, but directly depend on the availability of hardware descriptions [28], [29]. Moreover, for GPUs, their structural complexity and transistor density restrict the adoption to specific evaluations (e.g., on functional units [30], or controllers [31]). For instance, the RT level analysis of a GPU on a CNN layer might require up to 10,000 days! [32]. The functional evaluations are slightly more efficient in performance and can support the characterization of the software/hardware interaction of a system [33], but are restricted to data path structures (e.g., memories and functional units). Figure 1 depicts the corruption patterns on a GEMM operation obtained through structural analyses of faulty TCUs inside a GPU. Unfortunately, these analyses might introduce inaccuracies in the resilience evaluations [34].

5) ***physical evaluations****:* resort to evaluations (e.g., beam experiments) of the ML model's architecture on real devices on special facilities to evaluate the system's reliability. These evaluations are supremely effective for the overall system evaluation but might hardly provide fine-grain evaluations on specific structures of a hardware accelerator.

6) ***hybrid mechanisms****:* (a.k.a. cross-layer or multi-level) combine two or more strategies to provide an efficient trade-off between evaluation accuracy, and performance to determine vulnerable structures in ML hardware and the impact on ML applications. These strategies allow the evaluation of some hardware structures that cannot be covered by other

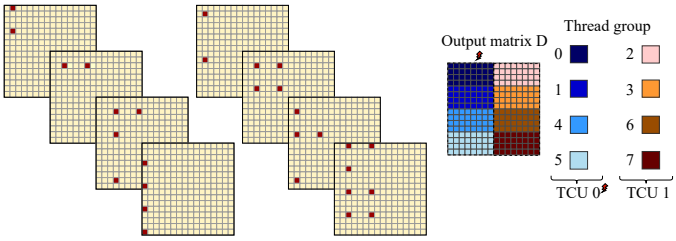

Fig. 1: Corruption patterns in GEMM outputs from faults in TCUs [33].

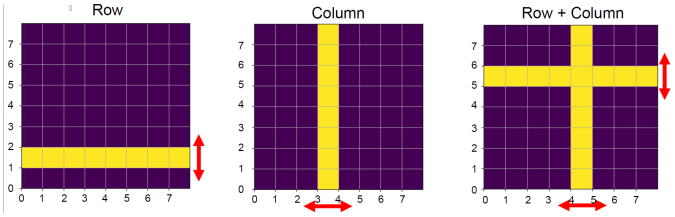

Fig. 2: Some corruption patterns in the GEMM's output by faults in a scheduler controller [32].

strategies (e.g., faults in controllers, functional units, and schedulers [27], [32], [35], [36]). However, the integration of hybrid strategies for evaluation require considerable engineering effort and the description of efficient mechanisms (error functions, tables, corruption masks) for the interaction among the analyzed abstraction levels.

B. Reliability vulnerabilities and mitigation opportunities in GPUs

Memory vulnerabilities in the first GPUs were mostly corrected by Error Correction Codes (ECCs), error containment mechanisms, memory row remapping, and dynamic page offlining for main memories and registers files [37]. Unfortunately, these protection schemes do not fit all GPUs structures, such as temporal near-register files or buffers, and local memories, leaving open the possibility of silent corruption effects on some of the threads of an application.

Functional units and special cores in GPUs, integer, FPU, and TCU cores are massively reused during the execution of ML workloads to compute fragments of large GEMM operations [38]. Consequently, a faulty unit may produce one or several silent computation errors in the results. In [36], several spatial corruption patterns by faults are identified in the instructions on the output feature maps of convolutional layers. In [33], scalar and spatial corruptions from faults are identified in the TCU's structure, as depicted in Figure 1.

Finally, in crucial structures in GPUs, such as controllers and scheduler units, a faulty unit might corrupt and collapse the system's/application's operation. In most cases, errors from GPU schedulers and controllers cause hanging and crashing effects. However, several reliability assessments [27] indicate that some faults in controllers might propagate silently and cause errors in application results. Similarly, several corruption patterns are identified in results from faults in a GPU controller [32], as depicted in Figure 2.

Since resilience evaluations support the identification of vulnerable structures in large hardware accelerators, including GPUs, they also contribute to determine opportunities to increase fault tolerance. At application level, mitigation strategies mostly focus on algorithmic optimizations to increase reliability, such as ABFT [10], [39], and compression schemes.

At structural level, a reliability assessment on GPUs indicates that the scheduling policies in hardware controllers, to access and use the available parallel units (SMs, FPUs, INTs, and TCUs), impact the generation of error corruptions on ML workloads. Thus, the clever management of the policies provides an opportunity to reduce the fault vulnerability of a system. Figure 3 depicts the corruption effects on GEMM outputs from faulty TCUs on the same GPU with different scheduling policies [40].

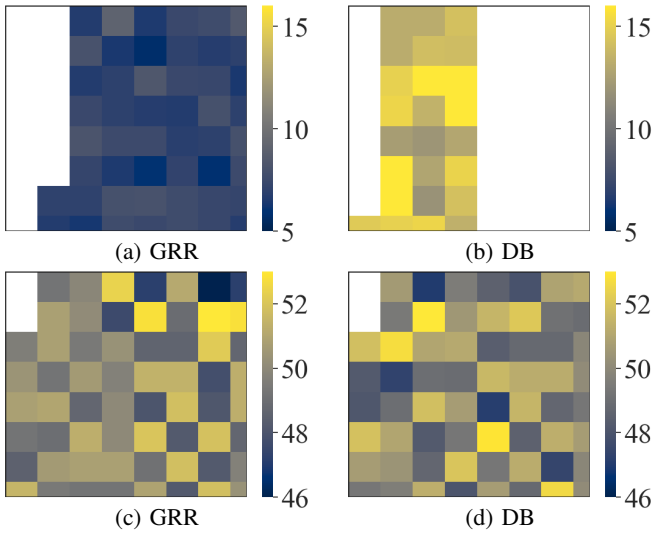

Fig. 3: Accumulated number of corruptions in the GEMM's output by a faulty TCU in a GPU with 7 SMs and 28 TCUs (*Top*) and a GPU with 2 SMs and 4 TCUs *(Bottom)*, both GPUs under two scheduling policies: *Global Round Robin* (GRR) and *Distributed Block* (DB). As depicted, the core organization of a device and the scheduling policy impacts the corruption effects on GEMM operations.

In memories, ECC and spare hardware are the principal mechanisms adopted in commercial devices for mitigation purposes. Similarly, redundancy and spare mechanisms for parallel functional units were proposed for GPUs [41]. In addition, several works proposed and analyzed the impact of trans-/mixed-precision solutions to increase the reliability of ML applications with promising results, during the training and inference stages [42]. Other mitigation analyses and strategies resort to approximate computing [43], quantization strategies on ML workloads [44], and the use of emerging number formats and hardware [33], [45], [46]. Furthermore, less invasive strategies resort to software-based solutions to partially schedule workloads in GPUs, as well as software-

TABLE I: ECC efficiency for different models on two NVIDIA GPUs, and theoretical efficiency of a Selective ECC.

		ECC On/Off Ratio		Critical SDC (%)	
		SDC	DUE	ECC Off	ECC On
Kepler GPU	YoloV1	0.2	1.1	8.0%	61.1%
	Faster R-CNN	0.1	1.4	5.5%	25.0%
	ResNet 200	0.2	1.5	8.0%	15.5%
Ampere GPU	ViT B16-224	0.4	1.5	7.7%	2.4%
	ViT B16-384	0.3	1.6	8.0%	3.9%
	ViT L14-224	0.2	2.4	37.3%	10.0%
Theoretical Selective ECC*	LeNet 5	*N/A*	*N/A*	1.77%	0.03%
	ResNet 18	*N/A*	*N/A*	8.44%	0.01%

*Theoretical selective ECC data only consider errors in the memory

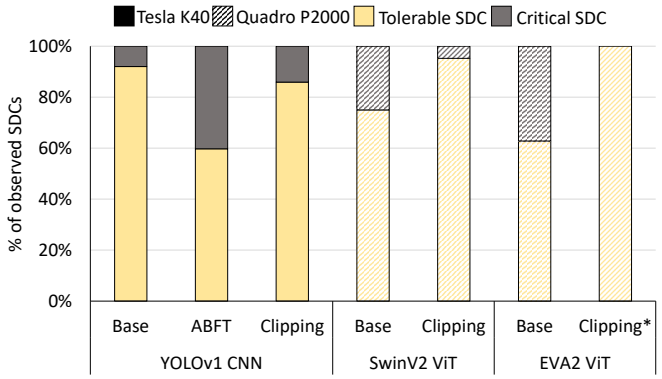

Fig. 4: ABFT and Value Clipping efficiency when employed on YOLOv1, SwinV2 ViT, and EVA2 ViT.

based hardening solutions at the scalar level to reduce the impact of large-magnitude errors [47].

In the remainder of this section, we describe our experiments using neutron beams to determine GPU failure rate and identify potential weaknesses that could compromise DNN reliability. Our findings allowed us to propose effective hardening techniques that significantly reduce DNN misprediction rates.

C. Characterizing and Mitigating Radiation-induced Faults

We have chosen a set of representative DNN models to evaluate the effect of radiation-induced transient faults. These models include YOLOv1 and Faster R-CNN for object detection [48], [49], ResNet [50] and LeNet [51] for image classification, and Large Vision Transformers (ViTs) for image classification [52], [53].

1) ***Experimental setup****:* experimental tests were conducted at ChipIR in Rutherford Appleton Laboratory (RAL, UK) and LANSCE in Los Alamos National Laboratory (LANL, US). Both facilities provide a neutron beam to simulate atmospheric neutron effects in electronic devices. This allows for the measurement of realistic failure rates of the device while executing a code. These experiments measured the probability of a neutron causing a failure in the GPU. The failure rate calculated in these experiments can be used to estimate the terrestrial failure rate caused by neutrons on a GPU. For a detailed description, please refer to [10], [54], and [55].

We continuously run the code in the irradiated GPU with a known input, checking that the application output is correct. Any expected and produced output mismatches are classified as Silent Data Corruptions (SDC). As the DNNs are approximate methods, we classify the SDCs into two categories: (1) Tolerable SDCs, which do not change the inference even if the output tensors differ; (2) Critical SDCs, the classification or detection is changed. Finally, if the application crashes or the operating system stops responding, it is a Detected Unrecoverable Error (DUE). We have extracted data from neutron beam experiments conducted on three NVIDIA GPUs, namely Kepler (Tesla 40), Pascal (Quadro P2000), and Ampere (Quadro RTX A2000).

2) ***ECC Efficacy on DNNs****:* modern GPUs are equipped with Single Error Correction Double Error Detection (SECDED) ECC, which helps to enhance their reliability while maintaining high performance [56]–[58]. Table I presents the ratio of the observed failure rate for SDCs and DUEs ($(ECC\ On\ rate)/(ECC\ Off\ rate)$) and the percentage of the observed Critical SDCs for two NVIDIA GPU architectures. The table also presents the results of a selective ECC approach we proposed that can improve the error correction efficiency of DNNs by targeting only the parameters that lead to critical SDCs [54], [59].

While the ECC can reduce the SDC failure rate by an order of magnitude, the ECC can reduce neither the number of critical nor multiple errors that come from unprotected GPU units. Even with ECC protection, we show that the critical error rate is not neglectable, on average 33.9% for Kepler GPU and 5.4% for Ampere GPU. Contrarily, the Selective ECC that protects only the necessary parameters of the DNN, focusing on the Critical SDC, can reduce the Critical SDCs to 0.03% for LeNet 5 and 0.01% for ResNet 18. ECC is a fault tolerance method for memories that neglects the high number of computation units necessary to compute the large modern DNNs. In the following subsection, we present how the computation operations can be hardened by using simple approaches, i.e., by applying ABFT on GeMM kernels and value clipping on the output of the layers.

3) ***Hardening based on ABFT and Value Clipping****:* while the GPU executes the DNN, a transient fault changes the circuit's expected value. The fault propagates through the software instructions until it reaches the output tensor produced by a layer. The fault can corrupt the DNN internal tensors in different ways, e.g., single or multiple values within a tensor, an entire row or column (often referred as line), or a block of values (details at Section II-A). These errors propagate through the network and eventually reach the last layer of the DNN, which can produce an incorrect inference. As ECC protect only errors that affect memory values, recent works have leveraged GeMM-based protection methods applied for DNNs, i.e., ABFT for GeMM [10], [58], [60].

As large machine learning models become increasingly prevalent in safety-critical applications, ABFT applied to

GeMM presents some limitations [58]. Due to the high number of memory accesses required by ABFT, its implementation can add more than 60% overhead for large DNNs [58]. Additionally, recent works have demonstrated that radiation-induced faults can drastically change the magnitude of the internal layer values. Since the floating-point representation range is large [10], [55], [61]–[65], the corrupted values can go to *infinity* or even become *Not a Number* (NaN). In order to increase the reliability of DNNs, we propose to not propagate the corrupted values, but instead clip them to *acceptable values*. On YOLOv1, we modified the MaxPooling layers to propagate the value inside a range – $10\times$ the largest value of the given MaxPooling layer – instead of the largest value. For ViTs, we modified the Identity layers to clip corrupted values to $1.3\times$ the maximum value in a given Identity layer.

We measured the efficiency of both ABFT and Value Clipping in protecting against neutron-induced faults for DNNs on GPUs. Figure 4 shows the percentages of Critical and Tolerable SDCs observed on beam experiments for each fault tolerance tested on YOLOv1 and two Large ViT models. YOLOv1 runs on an NVIDIA Tesla K40 (Kepler GPU), and the SwinV2 and EVA2 ViTs run on an NVIDIA Quadro P2000 (Pascal GPU). As Kepler GPU has native ECC, the Base version considers only the DNN running with only the ECC enabled without software protection. For Pascal GPU, the Base version is the ECC-disabled version. ABFT can correct about 60% of the SDCs. If only the critical SDCs are considered, an ABFT-protected YOLOv1 is more resilient than an ECC-protected version. This is because ABFT corrects all the detected errors that affect GEMM computation.

For all cases, value clipping techniques applied to YOLOv1, SwinV2, and EVA2 lead to a lower percentage of Critical SDcs than the base DNN. Value clipping helps reduce the Critical SDCs percentages to 14.1% for YOLOv1 and 4.8% for SwinV2. On EVA2, protected by clipping, no Critical SDCs were observed. For EVA2, if we consider the error bars based on the results of neutron beam experiments (calculated using Quinn and Tompkins approach for zero failures [66]), we can expect a maximum of 31% of critical SDCs when the model is protected by clipping. As a comparison, the critical SDCs percentage of unprotected EVA2 is 37.2%

III. Remote Side-Channel Attacks on FPGA-based Neural Networks Accelerators and Possible Countermeasures

ML circuits implemented in FPGA logic, especially in multi-tenant situations, are susceptible to attack. In this section, we focus on side-channel attacks in which adversaries attempt to obtain information about the FPGA-implemented ML circuit using circuitry embedded in the device.

In remote cloud FPGAs, the most straightforward way to snoop on unsuspecting victim circuits is to observe small drops in supply voltage due to both resistive and inductive drops in the Power Distribution Network (PDN). Since the signal delay in digital circuits varies as supply voltage changes, custom circuits can be used to sense on-chip voltage changes. For FPGAs, these circuits are often implemented as Ring Oscillators (ROs) [67] or Time-to-Digital Converters (TDCs) [68]. ROs, which are typically based on an inverter chain with an odd number of inverters, are simple to implement but require significant time periods for voltage estimation. Alternatively, TDCs [67] measure circuit delay within a clock cycle by determining the distance through a tapped delay line a signal can travel during the cycle [68]. This characteristic makes TDCs effective in obtaining side-channel information. Compared to an RO sensor, a TDC sensor needs careful placement and calibration to ensure its delay is matched to the clock period, or else its output value can saturate. The high-speed carry logic in modern FPGAs makes a suitable delay line with taps that are on the order of picoseconds, allowing for accurate voltage change estimations following sensor calibration [69].

TDCs have been used to collect side-channel information from a wide variety of ML circuits implemented in FPGAs. Tian et al. [70] implemented a remote power attack that extracts details of a neural network accelerator implemented in an FPGA. The accelerator uses a domain-specific instruction set architecture to implement ML algorithms. The TDC tracks voltage fluctuations which helps the adversary identify groups and parameters for executed accelerator instructions. This work was later expanded [71] to identify power consumption signatures involving the FPGA's external bus interface and system DRAM. Crucially, voltage change patterns related to interface use and the accelerators can be used to trigger an attack. This trigger eliminates the need for external signal monitoring to launch an attack on a victim circuit. A more recent TDC-based approach [72] extracts parameters from folded neural network layers. Folded implementations reduce circuit area and power consumption by limiting the instantiation of layer hardware in the FPGA. Although tracking voltage changes in this environment is more challenging, folded parameters can be successfully extracted. This information can also be used to deduct neuron count in the ML circuit.

Several research projects have examined the extraction of images input into an FPGA-based ML circuit. In Huegle et al. [15], small voltage fluctuations determined by a TDC are used to recreate pixels. A generative CNN is used to analyze these fluctuations following training. Several datasets with greyscale images are used for analysis, and images are recreated with greater than 90% accuracy in most cases under a variety of operating conditions.

In the next subsection, we examine the extraction of greyscale images that are input into an FPGA-based ML circuit. Histograms derived from TDC-detected voltage changes [73] are used to reconstruct the input image. This approach has been validated on a commercial cloud platform, the Amazon Web Services (AWS) EC2 F1 [4]

A. TDC-Based Image Extraction Architecture

Our example considers a multi-layer CNN that takes in MNIST greyscale images with 28×28 eight-bit pixel values as input. As shown in Figure 5, the outputs of each CNN layer

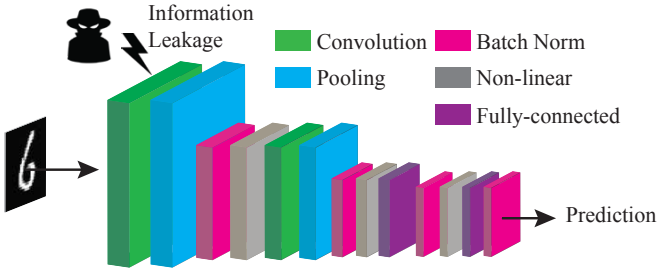

Fig. 5: Overview of steps in the CNN used for image extraction [73].

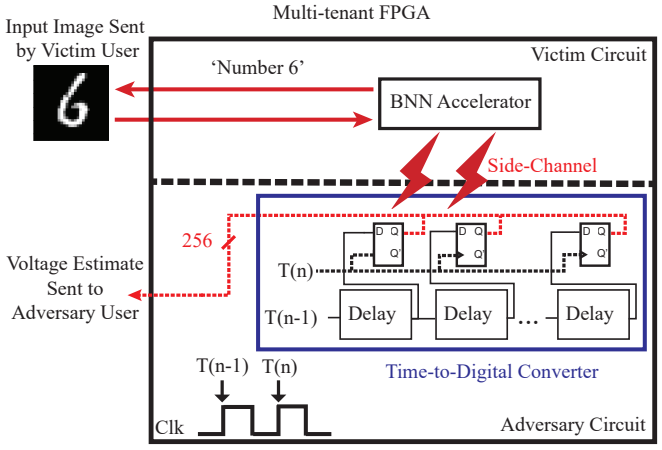

Fig. 6: Overview of ML attack implementation. The TDC outputs voltage estimates for each clock cycle of the first layer. These estimates are used to reconstruct the input image.

(feature maps) are passed as input to the next layer. Layers in the CNN include a non-linear function (creating complex input-output mappings), pooling (reducing the dimensionality of input feature maps by different methods, e.g., max pooling), batch normalization (normalizing input feature maps to decrease their variance), fully-connected layers (where each element of an output feature map is calculated by point-wise multiplication between a whole input feature map and a kernel of the same size), and convolution layers. Our attack focuses on the operation of the first (convolution) layer, which is shown in green. Binarized neural networks (BNNs), representing each element of the convolution kernel as a single bit, are used.

In this work, the TDC measures delay changes as the first CNN layer (BNN accelerator) computation is performed (Figure 6). The data from the TDC is used by the adversary to estimate the voltage drop across the FPGA PDN during the execution of the convolution layer, as the BNN accelerator does the image classification. The acquired voltage estimates serve as a side channel that can be used to extract the victim's input image data. The recovered image approximates the input image by distinguishing between foreground and background pixels of the image.

In the first convolution layer, an image is convolved with multiple distinct kernels to generate multiple output feature

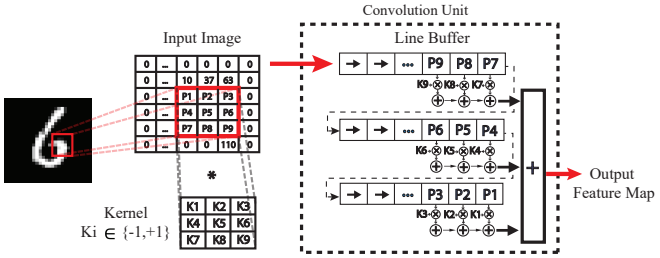

Fig. 7: Detailed view of the convolution unit. Output is generated from the 3×3 input image, shown in the red box, and the kernel.

maps. In our attack, we use a voltage estimate trace from the execution of the first kernel of the first convolution layer for an input image. Since we assume that the same image is evaluated by the FPGA accelerator multiple times, multiple (N) similar traces are collected using the same input image. After collecting multiple traces, the adversary takes the mean of the data values in the traces to obtain a single average trace of the voltage estimates during the execution of the first kernel of the first convolution layer. A high-pass filter is then used to remove noise. We leverage the observation that the background and foreground pixels can then be distinguished by analyzing the different magnitudes of the voltage in a trace of measurements. This information can be represented by a histogram of instance counts of magnitude values in the filtered trace. Points in the histogram are used to label pixels as belonging to the image foreground or background based on the magnitude of their voltage measurement. An image-denoising filter is applied to this preliminary recovered image to improve clarity. The result of the analysis is a reconstructed image that approximates the input image to the BNN.

The convolution unit uses a line buffer architecture to hold and provide data values to the convolution. As shown by the line buffer at the right in Figure 7, the line buffer is arranged in three rows, each of which processes one line of the convolution operation. The line buffer is a shift register that receives one pixel from the input feature map (the image) per clock cycle and shifts its values to the right. The length of each row in the line buffer matches the length of the input feature map of the convolution operation (28 for the first layer in our implementation). The rightmost word of each row of the line buffer enters the next row from the left, and the rightmost word of the last row is discarded. The rightmost three words of each of the three rows of the line buffer constitute the image window whose values are multiplied with values from the 3×3 kernel. Since binary kernels are used in a BNN, each image pixel in the current image window is added to or subtracted from (based on a kernel value of +1 or -1) the other pixels in one clock cycle using a combinational adder tree. One output feature map value is generated every clock cycle.

An adversary can take advantage of the shared FPGA PDN to sense local supply voltage changes, which can reveal information about the per-cycle power consumption in the convolution unit. The power consumption is due in part to

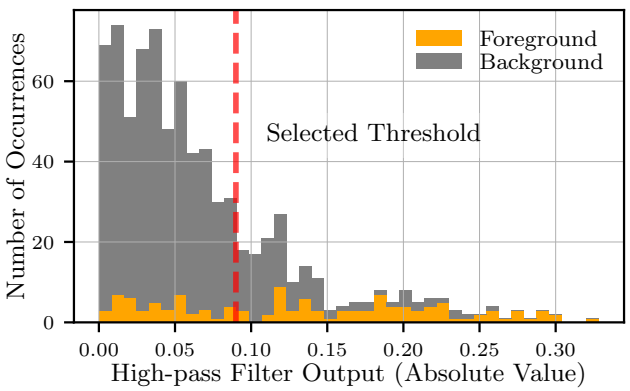

Fig. 8: Average of TDC voltage drop traces averaged over 6,000 runs on AWS F1. Histogram of the filtered TDC trace and the selected threshold.

the switching activity in the BNN accelerator, including the convolution unit, which causes supply voltage to be correlated to the data processed (larger magnitude data values lead to increased switching). The small PDN fluctuations are reflected in the sampled values of the TDC, and the TDC samples are then used to recover a facsimile of the input image.

TDC-Based Image Extraction Results: the attack architecture was implemented on AWS F1 instances. The AWS virtual machine (VM) is able to send input images to the FPGA and read TDC outputs from the FPGA, using built-in `peek()` and `poke()` functions. The TDC modules are physically separated from the BNN Accelerator without any direct communication. Since AWS F1 instance FPGAs currently only support use by a single customer at a time, this setup approximates a multi-tenant scenario.

To illustrate the range of voltage changes due to the convolution of the input image, a histogram of the absolute value of voltage drop measurements following high pass filtering is shown in Figure 8. The histogram contains 40 bins evenly distributed in value. The boundary between foreground and background pixels can be distinguished with a threshold. Generally, the processing of background pixels leads to small voltage drops, clustered on the left of the histogram, and the processing of foreground pixels leads to a range of larger voltage drops, on the right side of the histogram. The threshold can be identified by locating a downward gradient in occurrence counts over multiple bins. Figure 9 shows an original image and reconstructed images with and without denoising.

Countermeasures are needed to reduce the effectiveness of on-chip voltage measurement attacks. The extraction of voltage estimates could be impeded by the significant circuit switching of interfaces or other design components in the proximity of the convolution unit (e.g., active fences [74], [75]). Additionally, the pixel order of convolution unit processing could be scrambled on a per-image basis to make image reconstruction more difficult.

(a) (b) (c)

Fig. 9: (a) Input image, (b) The recovered image of same-SLR, experiment on AWS F1 for 6,000 runs - w/o denoising, (c) same as (b) with denoising

B. Training-Based Hiding Countermeasures against Input Recovery Side-Channel Attacks

Our work introduces an innovative training-based approach to reduce the impact of this threat of remote side-channel attacks. We train the classifier of the potential victim that needs to be protected in such a way that the classification task can still be performed as intended, as well as reduce the observable side-channel leakage. For that, we first replicate the input side-channel attack shown by Huegle et al. in *Power2Picture* [15], using the Zynq UltraScale+ MPSoC platform. Then, we modify the training algorithm for the victim network to account for the leakage that is used in that side-channel attack. We validate the feasibility of this countermeasure by performing the side-channel attack again and comparing the attack's results for both the original and protected network.

1) ***Methodology:*** in the proposed approach, we adapt the training of the victim classifier to fulfill two training objectives at the same time: (i) Perform the original image classification as intended, meaning that the model is able to solve the underlying problem with high accuracy. (ii) Integrate the model's leakage as loss into the training process, minimizing the victim classifier's leakage through the side-channel. Since the exact way of how the input and the weights of the victim network influence the measured trace is not known in our scenario, we cannot directly optimize the weights of the victim network to minimize the leakage. In technical terms, we cannot back-propagate through the side-channel, since it is unknown if this function is differentiable.

Our solution to address this problem is to implement and train a *surrogate model*, which gets the image and weights of the victim network as input, and output the trace this configuration would produce on the real FPGA. This model is differentiable and, thus, closes the gap between weights/images and the resulting power trace. For the training of the victim classifier, we first pass an image to the classifier and get its result. Then we take the original image and the weights of the classifier and pass them as input to the surrogate model, which then outputs the corresponding power trace.

Our objective is to force a constant output of the surrogate model for all inputs we use. For example, this can be either an all-zero output, or the average of all traces. The surrogate model itself is trained beforehand and is not modified when retraining the victim network to minimize its leakage. In fact,

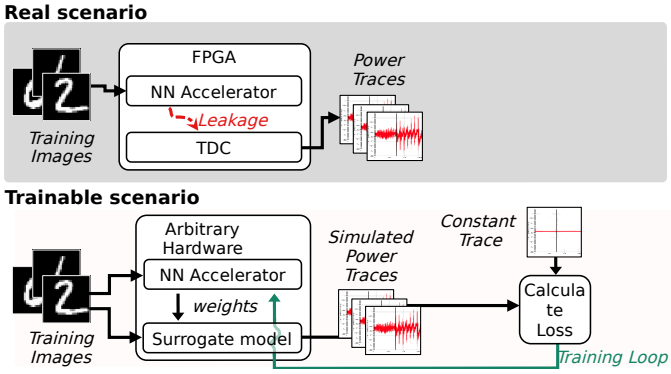

Fig. 10: A comparison between the real world scenario and our trainable simulation. Instead of using the side-channel, our surrogate model is used. The surrogate model was trained earlier, thus only the weights of the accelerator receive updates.

since our objective is to update the weights of the victim network and these weights are part of the input for the surrogate model, we optimize for input and not for weights in the surrogate model. The loss then gets combined with the loss of the image classification. An illustration of the setup is shown in Figure 10.

In summary, our method is implemented as follows: (i) Generate a training data set consisting of a large variety of image/weights combinations and their real world trace using the unprotected victim classifier. (ii) Train a surrogate model that outputs a trace given an image and the weights of the victim network. (iii) With the help of the surrogate model, train the victim network to perform the original classification task, while having minimized information leakage.

To support this setup, opposed to running the FINN NN Accelerator in *const* memory mode, as also presented in [15], we operate it in *decoupled* mode. In that mode, the weights can be changed at runtime without reprogramming the entire design. Overall, we use the same type of board (Xilinx Ultrascale+ ZCU104 Board), and similar neural network. Next to our countermeasure results, we report results for both the const and decoupled setup.

*2) **Result Metrics**:* to evaluate the effectiveness of the attack, we use result metrics as follows:

Reclassification Accuracy: As mentioned before, the test output of the generator consists of images recovered from the traces of the test data set. We can feed these images once more through the original (victim) classifier and measure the portion of images that were classified correctly. We call this metric *Reclassification Accuracy*. The closer the recovered images are to the original images, the more likely the classifier will classify them correctly. Thus, a higher reclassification accuracy is indicative of a more successful attack.

Average Pixel-Level Distance (APLD): To calculate the APLD, every recovered image is compared to its original by computing the average pixel-wise difference. The more similar the recovered images and the corresponding originals are, the smaller the pixel-level distance is. Ideally, for an attacker, there is no difference, resulting in a pixel-level distance of zero.

TABLE II: Reclassification accuracy, Mean Structural Similarity Index (MSSIM) and Average Pixel-Level Distance (APLD) for an accelerator without and with applied countermeasure.

model	attack success metrics		
	reclassification accuracy	MSSIM	APLD
ideally secure	10% (guessing)	close to 0.0	close to 255
perfect attack	100%	1.0	0.0
const (baseline)	65.75%	0.56	26.73
decoupled	38.70% (*-29.8%*)	0.34 (*-39.3%*)	32.19 (*+20.4%*)
countermeasure	22.07% (*-66.4%*)	0.25 (*-55.4%*)	35.15 (*+31.5%*)

Mean Structural Similarity Index (MSSIM): The MSSIM was developed by Wang et al. [76] and is a metric to evaluate the preservation of structural information in two images. It follows directly from the definition of the *Structural Similarity Index* (SSIM). Since our original images are highly structured, this metric provides useful insight into the quality of our results. The SSIM compares three aspects: luminance, contrast, and structure of both images. The MSSIM is defined by computing the SSIM multiple times. Here, the SSIM is not calculated for the entire image at once. In fact a sliding window is moved over the image, calculating the SSIM in every step. These values get averaged afterwards. For our results, we use a window size of 11. To get a single value for the data set, we simply average all MSSIMs pair-wise.

*3) **Results and discussion**:* overall, the countermeasure has shown to reduce the amount of information that is recoverable through the chosen side-channel, while maintaining a high classification accuracy. The results are summarized in Table II. The unprotected baseline approach with *const* memory mode reaches a reclassification accuracy of 65.75%. By applying the decoupled mode, we can already achieve results below the baseline, which is due to the additional streaming of the weights hiding some of the leakage. For our training-based countermeasure the *reclassification accuracy* dropped significantly down to only 22.07%. This drop means that the adversary could not restore the images with the same accuracy, indicating that our protection countermeasure effectively reduces the attack feasibility. This is also confirmed by the trend of the *MSSIM* and *APLD* metrics. Furthermore, it is remarkable that the initial classification accuracy of the protected neural network has only suffered a minimal loss with less than 1% accuracy degradation to the unprotected model. This is very important since we do not want our countermeasure to largely affect the primary task of the accelerator.

Overall, our countermeasure has shown to reduce the attack success by about 43%. With the countermeasure, the victim is able to improve its privacy by making the attack more difficult for the attacker with negligible losses in classification accuracy. As per our design, the countermeasure does not need additional hardware resources or runtime for inference,

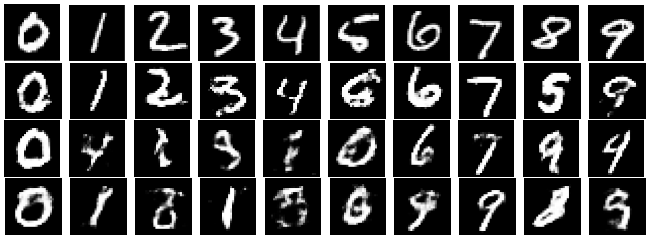

Fig. 11: Examples of (a) Original MNIST images, (b) recovered images in *const* memory mode (c) recovered images in *decoupled* memory mode, and (d) recovered images with the countermeasure applied.

since we only adapt the weights of the network. However, we were not able to completely prevent information being leaked through the side-channel. Besides the need for more research on this specific topic, our countermeasure is a complement in an existing security concept and not a standalone solution to prevent side-channel leakage.

IV. Conclusion

Applications of artificial intelligence continue to grow in importance and complexity. Contemporary AI applications typically require power-hungry compute acceleration to achieve expected performance. Unfortunately, the use of accelerators opens up the possibility of faults in the silicon and malicious attacks. In this paper, we have described these threats and associated countermeasures through a series of experimentation-based case studies.

We started our discussion in this paper by examining threats to AI computation on GPUs. More specifically, the impact of transient faults on device reliability is explored through a series of evaluations. These faults may be induced by radiation. Hardening can be achieved using algorithm-based fault tolerance and value clipping to mitigate the impact of radiation-induced faults. In memories, error-correcting codes and hardware redundancy provide protection.

In the second part of the paper, we examine the vulnerability of FPGA-based ML circuit inputs. We first outline a practical attack that allows for direct reconstruction of input images. This effort is supported by a time-to-digital converter-based sensor that can detect small voltage fluctuations during circuit operation. These fluctuations allow for image reconstruction one pixel at a time. We then describe an effective countermeasure that uses parameter training in an effort to prevent these fluctuations. Parameter classification is performed prior to circuit execution. The instrumentation of these parameters prevents side channel leakage when the ML circuit is executed.

As AI hardware continues to evolve, increased fault tolerance and security will be musts. New devices must be designed to withstand faults and attacks. This paper provides insights into several promising directions for this effort.

Acknowledgment

This work has been supported by the National Resilience and Recovery Plan (PNRR) through the National Center for HPC, Big Data and Quantum Computing. This research was partially funded by National Science Foundation grant CNS-1902532 and by the ANR-21-CE24-0015 "RE-TRUSTING" project.

References

[1] B. Dally, "Hardware for deep learning," in *HCS*, Aug 2023, pp. 1–58.
[2] J. Choquette, "NVIDIA Hopper H100 GPU: Scaling performance," *IEEE Micro*, vol. 43, no. 3, pp. 9–17, 2023.
[3] E. Nurvitadhi *et al.*, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC," in *FPT*, 2016, pp. 77–84.
[4] Amazon.com, Inc., "Amazon AWS F1," https://aws.amazon.com/ec2/instance-types/f1, Accessed: Apr. 2024.
[5] Google Cloud. (2023) AI and machine learning solutions. [Online]. Available: https://cloud.google.com/solutions/ai
[6] Amazon Web Services. (2023) Machine learning and artificial intelligence. [Online]. Available: https://aws.amazon.com/machine-learning
[7] Microsoft Azure. (2023) Azure machine learning - ML as a service. [Online]. Available: https://azure.microsoft.com/en-us/products/machine-learning
[8] Y. Ibrahim *et al.*, "Soft error resilience of deep residual networks for object recognition," *IEEE Access*, vol. 8, pp. 19 490–19 503, 2020.
[9] D. A. G. Goncalves de Oliveira *et al.*, "Evaluation and mitigation of radiation-induced soft errors in graphics processing units," *IEEE Trans. Computers*, vol. 65, no. 3, pp. 791–804, 2016.
[10] F. F. d. Santos *et al.*, "Analyzing and increasing the reliability of convolutional neural networks on GPUs," *IEEE Trans. Reliability*, vol. 68, no. 2, pp. 663–677, 2019.
[11] P. Rech *et al.*, "Impact of GPUs parallelism management on safety-critical and HPC applications reliability," in *DSN*, 2014, pp. 455–466.
[12] M. Traiola *et al.*, "Impact of high-level-synthesis on reliability of artificial neural network hardware accelerators," *IEEE Trans. Nuclear Science*, pp. 1–1, 2024.
[13] S. Moini *et al.*, "Understanding and comparing the capabilities of on-chip voltage sensors against remote power attacks on FPGAs," in *MWSCAS*, 2020, pp. 941–944.
[14] M. Zhao *et al.*, "FPGA-based remote power side-channel attacks," in *SP*, 2018, pp. 229–244.
[15] L. Huegle *et al.*, "Power2Picture: Using generative CNNs for input recovery of neural network accelerators through power side-channels on FPGAs," in *FCCM*, 2023, pp. 155–161.
[16] D. Spielmann *et al.*, "RDS: FPGA routing delay sensors for effective remote power analysis attacks," *IACR Tran. Cryptographic Hardware and Embedded Systems*, 2023.
[17] A. Singh *et al.*, "Silent data errors: Sources, detection, and modeling," in *VTS*, 2023, pp. 1–12.
[18] IEEE, "The international roadmap for devices and systems: 2022," in *Institute of Electrical and Electronics Engineers (IEEE)*, 2022.
[19] H.-G. Stratigopoulos *et al.*, "Testing and reliability of spiking neural networks: A review of the state-of-the-art," in *DFT*, 2023, pp. 1–8.
[20] F. Libano *et al.*, "Selective hardening for neural networks in FPGAs," *IEEE Trans. Nuclear Science*, vol. 66, no. 1, pp. 216–222, 2019.
[21] R. L. Rech Junior *et al.*, "High energy and thermal neutron sensitivity of Google Tensor Processing Units," *IEEE Trans. Nuclear Science*, vol. 69, no. 3, pp. 567–575, 2022.
[22] M. B. Sullivan *et al.*, "Characterizing and mitigating soft errors in GPU DRAM," in *MICRO*, 2021, p. 641–653.
[23] A. Ruospo *et al.*, "A survey on deep learning resilience assessment methodologies," *Computer*, vol. 56, no. 2, pp. 57–66, 2023.
[24] I. T. Bhatti *et al.*, "A formal approach to identifying the impact of noise on neural networks," *Commun. ACM*, vol. 65, no. 11, p. 70–73, oct 2022.
[25] B. Reagen *et al.*, "Ares: a framework for quantifying the resilience of deep neural networks," in *DAC*, 2018.
[26] O. Villa *et al.*, "PNVBit: A dynamic binary instrumentation framework for NVIDIA GPUs," in *MICRO*, 2019, p. 372–383.
[27] J. D. Guerrero Balaguera *et al.*, "Understanding the effects of permanent faults in GPU's parallelism management and control units," in *SC*, 2023.
[28] K. Andryc *et al.*, "FlexGrip: A soft GPGPU for FPGAs," in *FPT*, 2013, pp. 230–237.

[29] J. E. R. Condia *et al.*, "FlexGripPlus: An improved GPGPU model to support reliability analysis," *Microelectronics Reliability*, vol. 109, p. 113660, 2020.

[30] S. Azimi *et al.*, "Evaluation of transient errors in GPGPUs for safety critical applications: An effective simulation-based fault injection environment," *Journal of Systems Architecture*, vol. 75, pp. 95–106, 2017.

[31] J. E. R. Condia *et al.*, "Microarchitectural reliability evaluation of a block scheduling controller in gpus," in *ISVLSI*, 2022, pp. 26–31.

[32] F. F. d. Santos *et al.*, "Revealing GPUs vulnerabilities by combining register-transfer and software-level fault injection," in *DSN*, 2021, pp. 292–304.

[33] R. L. Sierra *et al.*, "Analyzing the impact of different real number formats on the structural reliability of TCUs in GPUs," in *VLSI-SoC*, 2023, pp. 1–6.

[34] G. Papadimitriou *et al.*, "Demystifying the system vulnerability stack: Transient fault effects across the layers," in *ISCA*, 2021, pp. 902–915.

[35] J. E. R. Condia *et al.*, "A multi-level approach to evaluate the impact of GPU permanent faults on CNN's reliability," in *ITC*, 2022, pp. 278–287.

[36] C. Bolchini *et al.*, "Fast and accurate error simulation for CNNs against soft errors," *IEEE Trans. Computers*, vol. 72, no. 4, pp. 984–997, 2023.

[37] Nvidia, "Nvidia gpu memory error management," 2024, url: https://docs.nvidia.com/deploy/a100-gpu-mem-error-mgmt/index.html.

[38] B. R. Boswell *et al.*, "Generalized acceleration of matrix multiply accumulate operations," Jul. 2 2019, US Patent 10,338,919.

[39] J. Kosaian *et al.*, "Arithmetic-intensity-guided fault tolerance for neural network inference on GPUs," in *SC*, 2021.

[40] R. Limas Sierra *et al.*, "Analyzing the impact of scheduling policies on the reliability of GPUs running CNN operations," in *VTS*, 2024, pp. 1–6.

[41] J. R. Nickolls, "Defect tolerant redundancy," Apr. 12 2005, US Patent 6,879,207.

[42] A. C. I. Malossi *et al.*, "The transprecision computing paradigm: Concept, design, and applications," in *DATE*, 2018, pp. 1105–1110.

[43] C.-Y. Chen *et al.*, "Exploiting approximate computing for deep learning acceleration," in *DATE*, 2018, pp. 821–826.

[44] N. Fasfous *et al.*, "Mind the scaling factors: Resilience analysis of quantized adversarially robust CNNs," in *DATE*, 2022, pp. 706–711.

[45] G. Gavarini *et al.*, "On the resilience of representative and novel data formats in CNNs," in *DFT*, 2023, pp. 1–6.

[46] R. Limas Sierra *et al.*, "Exploring hardware fault impacts on different real number representations of the structural resilience of tcus in gpus," *Electronics*, vol. 13, no. 3, 2024.

[47] G. Li *et al.*, "Understanding error propagation in GPGPU applications," in *SC*, Nov 2016, pp. 240–251.

[48] J. Redmon *et al.*, "You only look once: Unified, real-time object detection," in *CVPR*, June 2016.

[49] S. Ren *et al.*, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *NIPS*, 2015.

[50] K. He *et al.*, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[51] Y. Lecun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[52] Z. Liu *et al.*, "Swin transformer v2: Scaling up capacity and resolution," in *CVPR*, 2022, pp. 12 009–12 019.

[53] Y. Fang *et al.*, "Eva-02: A visual representation for Neon Genesis," *arxiv*, 2023.

[54] M. Traiola *et al.*, "harDNNing: a machine-learning-based framework for fault tolerance assessment and protection of DNNs," in *ETS*, 2023, pp. 1–6.

[55] L. Roquet *et al.*, "Cross-Layer Reliability Evaluation and Efficient Hardening of Large Vision Transformers Models," in *DAC*, Jun. 2024. [Online]. Available: https://hal.science/hal-04456702

[56] P. Rech *et al.*, "Measuring the radiation reliability of SRAM structures in GPUs designed for HPC," in *SELSE*, 2014.

[57] S. K. S. Hari *et al.*, "Low-cost program-level detectors for reducing silent data corruptions," in *DSN*, 2012, pp. 1–12.

[58] S. K. S. Hari *et al.*, "Making convolutions resilient via algorithm-based error detection techniques," *IEEE Trans. Dependable and Secure Computing*, pp. 1–1, 2021.

[59] M. Traiola *et al.*, "A machine-learning-guided framework for fault-tolerant dnns," in *2023 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2023, pp. 1–2.

[60] T. Marty *et al.*, "Enabling overclocking through algorithm-level error detection," in *FPT*, 2018, pp. 174–181.

[61] G. Li *et al.*, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *SC*, 2017, pp. 1–12.

[62] Z. Chen *et al.*, "A low-cost fault corrector for deep neural networks through range restriction," in *DSN*, 2021.

[63] L.-H. Hoang *et al.*, "FT-ClipAct: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," in *DATE*, 2020.

[64] N. Cavagnero *et al.*, "Transient-fault-aware design and training to enhance DNNs reliability with zero-overhead," in *IOLTS*, 2022.

[65] G. Gavarini *et al.*, "Evaluation and mitigation of faults affecting swin transformers," in *IOLTS*, 2023.

[66] H. Quinn *et al.*, "Measuring zero: Neutron testing of modern digital electronics," *IEEE Trans. Nuclear Science*, pp. 1–1, 2024.

[67] K. M. Zick *et al.*, "Sensing nanosecond-scale voltage attacks and natural transients in FPGAs," in *FPGA*, 2013, pp. 101–104.

[68] F. Schellenberg *et al.*, "An inside job: Remote power analysis attacks on FPGAs," in *DATE*, 2018, pp. 1111–1116.

[69] S. Moini *et al.*, "Voltage sensor implementations for remote power attacks on FPGAs," *ACM Trans. Reconfigurable Technology and Systems*, vol. 16, no. 1, pp. 1–21, Mar. 2023.

[70] S. Tian *et al.*, "Remote power attacks on the Versatile Tensor Accelerator in multi-tenant FPGAs," in *FCCM*, 2021, pp. 242–246.

[71] S. Tian *et al.*, "A practical remote power attack on machine learning accelerators in cloud FPGAs," in *DATE*, 2023, pp. 1–6.

[72] V. Meyers *et al.*, "Reverse engineering neural network folding with remote FPGA power analysis," in *FCCM*, 2022, pp. 1–10.

[73] S. Moini *et al.*, "Power side-channel attacks on BNN accelerators in remote FPGAs," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 2, pp. 357–370, 2021.

[74] J. Krautter *et al.*, "Active fences against voltage-based side channels in multi-tenant FPGAs," in *ICCAD*, 2019, pp. 1–8.

[75] O. Glamocanin *et al.*, "Active wire fences for multitenant FPGAs," in *DDECS*, 2023, pp. 1–8.

[76] Z. Wang *et al.*, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Processing*, vol. 13, pp. 600 – 612, 05 2004.